What is claimed is:

1. A method comprising:

configuring processors with multiple threads of execution to execute a critical section of code, the processors operable to execute the critical section in turns; and

controlling the threads of execution of the processors to avoid occurrence of idle time between execution of the critical section by the processors.

2. The method of claim 1 wherein controlling comprises:

enabling the threads of execution on each of the processors to execute in order via inter-thread signaling.

3. The method of claim 2 wherein enabling comprises:

enabling each thread, when such thread is executing, to execute a first instruction to cause an inter-thread signal to a next thread to be generated; and

enabling the thread to execute a second instruction, which causes the thread to wait for an inter-thread signal from a previous thread, only after a write latency required by the first instruction.

4. The method of claim 3 wherein the write latency comprises at least three instruction cycles.

5. The method of claim 3 wherein the processors each comprise a register through which an inter-thread signal to the next thread is given.

6. The method of claim 5 wherein the processors each comprise registers through which inter-thread, inter-processor signaling can occur.

7. The method of claim 6 wherein the processors use external registers to enable inter-thread, inter-processor signaling to occur.

8. The method of claim 2 wherein controlling further comprises:

enabling each thread of execution on each of the processors to relinquish control of a program comprising the critical section as soon as the critical section has been executed.

9. The method of claim 1 wherein controlling comprises:

enabling each thread of execution on each of the processors to relinquish control of the program comprising the critical section as soon as the critical section has been executed.

10. The method of claim 1 wherein the processors are operable to execute at least two of the critical sections of code.

11. The method of claim 10 wherein the processors comprise a functional pipeline in a network processor.

12. The method of claim 11 wherein one of the critical sections of code comprises a metering microblock and the another of the critical sections of code comprises a congestion avoidance microblock.

13. The method of claim 1 wherein the processors comprise a functional pipeline, and one or more of the critical sections of code are executed in the functional pipeline.

14. The method of 13 wherein one of the one or more critical sections of code comprises an Asynchronous Transfer Mode (ATM) receive processing microblock.

15. The method of claim 13 wherein one of the one or more critical sections comprises an ATM traffic management processing microblock.

16. A article comprising:

a storage medium having stored thereon instructions that when executed by a machine result in the following:

configuring processors with multiple threads of execution to execute a critical section of code, the processors operable to execute the critical section in turns; and

controlling the threads of execution of the processors to avoid occurrence of idle time between execution of the critical section by the processors.

17. The article of claim 16 wherein controlling comprises:

enabling the threads of execution on each of the processors to execute in order via inter-thread signaling.

18. The article of claim 17 wherein enabling comprises:

enabling each thread, when such thread is executing, to execute a first instruction to cause an inter-thread signal to a next thread to be generated; and

enabling the thread to execute a second instruction, which causes the thread to wait for an inter-thread signal from a previous thread, only after a write latency required by the first instruction.

19. The article of claim 16 wherein controlling comprises:

enabling each thread of execution on each of the processors to relinquish control of the program comprising the critical section as soon as the critical section has been executed.

20. A network processor comprising:

a processor; and

multi-threaded processors, having threads of execution, configurable by the processor to execute a critical section of code and operable to execute the critical section in turns; and

wherein the threads of execution of the multi-threaded processors are controllable to avoid occurrence of idle time between execution of the critical section by the multi-threaded processors.

21. The network processor of claim 20 wherein each thread, when such thread is executing, is controllable to execute a first instruction to cause an inter-thread signal to a next thread to be generated, and the thread is further controllable to execute a second instruction, which causes the thread to wait for an inter-thread signal from a previous thread, only after a write latency required by the first instruction.

22. A network processor of claim 20 wherein each thread of execution on each of the multi-threaded processors is controllable to relinquish control of the program comprising the critical section as soon as the critical section has been executed.

23. A system comprising:

a memory system to store a critical section of code;

processors, coupled to the memory system, having multiple threads of execution to execute the critical section of code, the processors operable to execute the critical section in turns; and

wherein the threads of execution of the processors are controllable to avoid occurrence of idle time between execution of the critical section by the processors.

24. The system of claim 23 wherein each thread, when such thread is executing, is controllable to execute a first instruction to cause an inter-thread signal to a next thread to be generated, and the thread is further controllable to execute a second instruction, which causes the thread to wait for an inter-thread signal from a previous thread, only after a write latency required by the first instruction.

25. The system of claim 23 wherein each thread of execution on each of the multi-threaded processors is controllable to relinquish control of the program comprising the critical section as soon as the critical section has been executed.